Meta-Learning via Hypernetworks

Author(s):
Zhao, Dominic; Kobayashi, Seijin; Sacramento, João; von Oswald, Johannes

Publication date:
2020-12

Permanent link:
https://doi.org/10.3929/ethz-b-000465883

Funding acknowledgement:
186027 - Probabilistic learning in deep cortical networks (SNF)

Meta-Learning via Hypernetworks

Dominic Zhao, Seijin Kobayashi
João Sacramento*, Johannes von Oswald*
* – equal contribution
Institute of Neuroinformatics, University of Zürich, ETH Zürich
{dozhao, seijink, rjoao, voswaldj}@ethz.ch

Abstract

Recent developments in few-shot learning have shown that during fast adaption, gradient-based meta-learners mostly rely on embedding features of powerful pre-trained networks. This leads us to research ways to effectively adapt features and utilize the meta-learner's full potential. Here, we demonstrate the effectiveness of hypernetworks in this context. We propose a soft weight-sharing hypernetwork architecture and show that training the hypernetwork with a variant of MAML is tightly linked to meta-learning a curvature matrix used to condition gradients during fast adaptation. We achieve similar results as state-of-art model-agnostic methods in the overparametrized case, while outperforming many MAML variants without using different optimization schemes in the compressive regime. Furthermore, we empirically show that hypernetworks do leverage the inner loop optimization for better adaptation, and analyse how they naturally try to learn the shared curvature of constructed tasks on a toy problem when using our proposed training algorithm.

1 Introduction

The ability to generalize previous knowledge and adapt quickly to novel environments has been subject of intense machine learning research in the past few years. One of the cornerstones of recent progress of meta-learning algorithms are gradient-based methods such as Model-Agnostic Meta-Learning (MAML) [1], which take the initial parameters of a model as its meta-parameters. MAML recreates few-shot learning scenarios and trains the meta-parameters directly on how well they can solve new tasks after a few gradient steps, see Section 2.1.

Recent work has shown that MAML is mostly learning general features rather than finding fast-adaptable weights deep inside its model. In fact, it was demonstrated that few-shot learning in the hidden layers of a network has little to no effect on the performance of MAML [2, 3]. Furthermore, powerful models trained on rich enough data and without explicit meta-learning were shown to outperform most gradient-based meta-learning methods [4]. This is consistent with huge models being few-shot learners without explicit training at all [5]. These previous findings point to the untapped potential of fast adaptation within the neural network as a promising area of improvement for such meta-learning methods.

One promising scalable approach is to separate the model into shared meta-parameters and context parameters [6]. Here, the context parameters are the only parameters updated in the inner loop. This way, the context parameters can learn task specific information and quickly adapt while the meta-parameters are used as general reusable features.

Other approaches attempt to explicitly modulate the inner-loop training procedure by meta-learning learning rates or factorised preconditioning matrices [7–9]. Here, the actual model stays untouched and additional parameters are learned only to modulate the gradient with respect to the model parameters while learning new tasks.

4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada.

Here, we provide new insights on how hypernetworks [10, 11] implicitly combine these two seemingly different approaches. When trained with a variant of MAML, we show that hypernetworks implicitly modulate the inner loop optimization and adapt hidden layer features in a task-dependent manner. More generally, we demonstrate that hypernetworks learn features that directly support fast adaptation without any hand-designed add-ons or optimization variants. We propose a specific soft weight-sharing hypernetwork architecture and show that it achieves state-of-the-art results compared to other gradient-based methods. Our method performs comparable with MAML even if the number of parameters is drastically compressed. Our main contributions are as follows:

- We demonstrate the effectiveness of hypernetworks for fast adaptation – both in the compressed and overparametrized regime.
- By proposing a simple soft weight-sharing hypernetwork architecture, we outperform most MAML variants without any explicit add-ons or optimization algorithm changes. Furthermore, we show empirically that hypernetworks can indeed learn useful inner-loop adaptation information and are not simply learning better network features.
- We show theoretically that in a simplified toy problem hypernetworks can learn to model the shared structure that underlies a family of tasks. Specifically, its parameters model a preconditioning matrix equal to the inverse of the tasks' shared curvature matrix.

2 Background and Related Work

Our algorithm is intimately related to MAML, in particular to two of its variants, namely CAVIA [6] and Meta-Curvature [7]. We next briefly reintroduce the MAML algorithm, while referring to Appendix A.4 for a recapitulation of Meta-Curvature.

2.1 Model-Agnostic Meta-Learning

In the supervised learning setting, MAML optimizes the initial parameters of a model θ by minimizing the validation loss obtained after a few gradient steps. To do so, the training data is separated into training $\mathcal{D}^{\text{train}}$ and validation $\mathcal{D}^{\text{val}}$ sets. The validation loss is evaluated after one or more steps of stochastic gradient descent with respect to the training tasks on the meta-parameters. For simplicity, here we consider the case where only one gradient step is taken.

We now describe one iteration of MAML. First, we sample a set of B tasks, $T_B = \{\mathcal{T}_b\}_{b=1}^{B}$, with $\mathcal{T}_b \sim p(\mathcal{T})$ for some task distribution $p(\mathcal{T})$. A task $\mathcal{T}$ is associated with a loss function $\mathcal{L}_{\mathcal{T}}$. For each task $\mathcal{T}_b \sim p(\mathcal{T})$ we sample training and validation datasets $\mathcal{D}_b^{\text{train}}$ and $\mathcal{D}_b^{\text{val}}$, respectively. Then, for each task $\mathcal{T}_b$, we obtain the adapted parameters θ_b:

$$\theta_b = \theta - \eta \nabla_\theta \frac{1}{|\mathcal{D}_b^{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_b^{\text{train}}} \mathcal{L}_{\mathcal{T}_b}\left(f_\theta(x), y\right). \tag{1}$$

The initialization parameters θ are then updated to minimize the cumulative validation loss. This loss is accumulated over the B tasks in T_B:

$$\theta \leftarrow \theta - \gamma \nabla_\theta \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{D}_b^{\text{val}}|} \sum_{(x,y)\in\mathcal{D}_b^{\text{val}}} \mathcal{L}_{\mathcal{T}_b}\left(f_{\theta_b}(x), y\right). \tag{2}$$

Above, η and γ denote the inner- and outer-loop learning rates, respectively.

2.2 Hypernetworks

For our approach, we draw direct inspiration from ideas related to *fast-and-slow weights* [10, 12, 13] and the more recently introduced *hypernetworks* [11, 14, 15]. One specific approach to implement these ideas is to express each layer as a learnable linear combination of templates. Since the templates are the same for each layer, the linear coefficients contain information on how the templates are shared across the model. Furthermore, it was shown that compression by using a small template bank is possible without affecting generalization [11].

More formally, assume that the parameters of a neural network represented as a vector of reals $w \in \mathbb{R}^{N_w}$ are partitioned into C chunks, $w = [w^{(1)}, \ldots, w^{(C)}]$. Each chunk $w^{(i)} \in \mathbb{R}^{N_c}$ has dimension $N_c = N_w/C$ (we assume N_w/C lies in $\mathbb{N}$). Instead of learning the weights w directly, we now define a set of template weight vectors, the columns of matrix $\theta = [t_{(1)} \ldots t_{(K)}]$ with $t_{(k)} \in \mathbb{R}^{N_c}$. We assign a different embedding vector $\alpha^{(i)} \in \mathbb{R}^K$ to each chunk i.

A linear combination then yields the weights of chunk i:

$$w^{(i)} := \sum_{k=1}^{K} \alpha_k^{(i)} \, t_{(k)} = \theta \, \alpha^{(i)}. \tag{3}$$

Note that θ is shared across the chunks, which results in a lower number of trainable parameters in the network if $N_c \, K + C \, K < N_w$. It is shown in [14] that the embedding parameters α learn to reuse the various $t_{(k)}$ across layers in deep neural networks and share feature information throughout multiple layers in the network after training.

More generally, a hypernetwork is any network that generates the weights of another network, given an input. Numerous research studies show the usefulness of hypernetworks and their respective variants in various areas such as visual-reasoning [16], continual learning [17, 18], transfer learning [19] and few-shot learning [3]. We point the reader to [20] for a broad analysis of hypernetworks and other multiplicative interactions within neural networks.

3 Meta-learning via Hypernetworks

Here, we investigate hypernetworks in the context of meta-learning. Differently from [21, 3], we focus on the hypernetwork's capability to implicitly modulate inner-loop optimization. More concretely, we suggest that the hypernetwork parameters θ play a role similar to the preconditioning matrix M introduced in the Meta-Curvature algorithm (reviewed in Appendix A.4).

Our meta-learning algorithm is a variant of CAVIA, which learns a general initialization for *both* task embeddings and hypernetwork parameters. Thus, the hypernetwork weights θ as well as an initialization for a task-specific embedding α are learned in the outer loop of our algorithm. Our algorithm is also closely related to the T-Net model [22], the essential difference being the sharing of a hypernetwork across multiple layers.

3.1 Soft Weight-Sharing Architecture

In the following, we present a simple linear, soft weight-sharing architecture for our hypernetwork and leave more complicated deep hypernetworks for future research. Each row of the generated network weight matrix will be a linear combination of hypernetwork templates. More precisely, we parameterize a given convolutional layer $W^l \in \mathbb{R}^{C_{\text{in}} \times C_{\text{out}} \times N_{\text{kern}} \times N_{\text{kern}}}$ with C_{in} input channels, C_{out} output channels and kernel size N_{kern} as follows: we construct for each input channel c a weight chunk $R_c^l \in \mathbb{R}^{1 \times C_{\text{out}} \times N_{\text{kern}} \times N_{\text{kern}}}$ as a linear combination of templates $t_{(1)}, \ldots, t_{(K)}$.

Conceptually, this can be seen as soft-sharing the rows of the parameter matrices across all layers. We express R_c^l for a specific input channel c at layer l as:

$$R_c^l = \sum_{k=1}^{K} \alpha_k^{(l,c)} \, t_{(k)}, \tag{4}$$

and finally concatenate each channel to create the layer weights:

$$W^l = [R_1^l, \ldots, R_{C_{\text{in}}}^l] \tag{5}$$

In this setting, α represents the task embedding and $\theta = (t_{(1)}, \ldots, t_{(K)})$ the hypernetwork weights.

We are now ready to present our algorithm. We denote the prediction generated for some input x when using task embedding α by $f_{\theta,\alpha}(x)$. In the inner loop, we first adapt the task embedding:

$$\alpha_b = \alpha - \eta \, \nabla_\alpha \, \frac{1}{|\mathcal{D}_b^{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_b^{\text{train}}} \mathcal{L}_{\mathcal{T}_b} \left(f_{\theta,\alpha}(x), y \right). \tag{6}$$

Then, in the outer-loop, we train the hypernetwork as well as an initialization for the embeddings using the validation loss, evaluated at the predictions generated using the adapted task embedding:

$$\theta \leftarrow \theta - \gamma \nabla_\theta \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{D}_b^{\text{val}}|} \sum_{(x,y) \in \mathcal{D}_b^{\text{val}}} \mathcal{L}_{\mathcal{T}_b} \left(f_{\theta, \alpha_b}(x), y \right). \tag{7}$$

$$\alpha \leftarrow \alpha - \gamma \nabla_\alpha \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{D}_b^{\text{val}}|} \sum_{(x,y) \in \mathcal{D}_b^{\text{val}}} \mathcal{L}_{\mathcal{T}_b} \left(f_{\theta, \alpha_b}(x), y \right) \tag{8}$$

We refer to a hypernetwork meta-learned in this fashion as a Meta-Hypernetwork (MH).

Our soft weight-sharing hypernetwork has the property that loss gradients $\nabla_\alpha \mathcal{L}$ with respect to the task embedding are *modulated* (multiplied) by θ, for some task loss $\mathcal{L} \equiv \mathcal{L}_{\mathcal{T}_b}$. This is most easily seen for a fully-connected layer l with weights $W^l = \theta\, A^l$ (of size $N_{\text{row}} \times N_{\text{col}}$), with hypernetwork weights $\theta \in \mathbb{R}^{N_{\text{row}} \times K}$ and embedding matrix $A^l \in \mathbb{R}^{K \times N_{\text{col}}}$. The gradient with respect to our embedding matrix is then $\nabla_{A^l} \mathcal{L} = \theta^\top \left(\frac{d}{dW^l} \mathcal{L} \right)^\top$. Thus, the hypernetwork weights θ can be seen as an implicit preconditioner of our task-specific embeddings, akin to the explicit preconditioner introduced in the Meta-Curvature algorithm (Appendix A.4).

4 Experiments

To show the effectiveness of the hypernetwork in achieving fast adaptation, we conduct experiments on standard few-shot regression and classification benchmarks.

4.1 Few-shot regression on Sinusoidal Task

Our first experiment follows the K-shot regression protocol outlined in [1, 8]. Given K input-output pairs $(x, f(x))$, with x uniformly sampled from $[-5, 5]$ and f a sinusoidal determined by random phase and amplitude in [0,π] and $[0.1, 5.0]$ respectively, the goal is to quickly learn $f(x)$ from these few examples.

Table 1: Few-shot regression loss measured by mean-squared error (MSE). Mean $\pm$ std. over 5 seeds.

Method	5-shot (↓)	10-shot (↓)
MAML [1]	$0.686^{\pm 0.070}$	$0.435^{\pm 0.039}$
LayerLR [7]	$0.528^{\pm 0.068}$	$0.269^{\pm 0.027}$
Meta-SGD [8]	$0.482^{\pm 0.061}$	$0.258^{\pm 0.026}$
MC2 [7]	$0.405^{\pm 0.048}$	$0.201^{\pm 0.020}$
MH	$0.501^{\pm 0.082}$	$0.281^{\pm 0.072}$

For fair comparison, we use the same neural network architecture proposed in [1], which consists of a fully-connected network with 2 hidden layers of size 40. For each of the rows of the hidden layers, we generate the weights with a linear combination of 40 templates. We use our Meta-Hypernetwork (MH) method to train the meta-learner. MH performs comparably with current state-of-the-art MAML-variants, see Table 1. Results from methods we compare to are taken from [7]. Additional details can be found in Appendix A.1.

4.2 Few-Shot Classification on MiniImageNet

To further demonstrate the effectiveness of our MH approach, we conduct experiments on the classic few-shot classification benchmark MiniImagenet [23]. For fair comparison, we again use the model proposed in [1] for all experiments, which consists of four convolutional layers with 64 filters, followed by a fully-connected layer. Each row of these convolutional layers is represented using one hypernetwork architecture described above. Note that therefore θ is shared across all convolutional layers in the network.

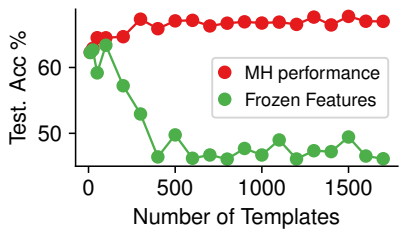

Figure 1: Difference in performance for 5-shot 5-way classification on MiniImageNet when adapting and not adapting the hypernetworks templates.

The number of templates of the hypernetwork K controls the expressiveness of the inner-loop gradient modulation. Our experiments show that the choice of K impacts the performance of our algorithm. In the following, we investigate three versions of MH with different number of parameters compared to the original model: compressed (MH-C), comparable (MH) and overparametrized (MH-O), with 50, 600 and 1500 templates respectively.

As expected, we observe a performance difference for our three variants in Table 2 in the 1-shot and 5-shot setting and show the effectiveness of our approach in all scenarios compared to similar gradient based methods using larger architectures or explicit gradient modulation.

4.3 Meta-hypernetworks leverage hidden layer capacity

To investigate the ability of MHs to shape inner-loop learning of its embeddings in the neural network hidden layers, we perform the experiments described in [2]. After training the hypernetwork, we freeze the layers during fast adaptation and only update the fully-connected layer (Frozen). This allows us to investigate the contribution of the meta-learned hypernetwork to the inner-loop optimization. Indeed, we observe (see Table 3) a dramatic effect on performance when disabling the training of hypernetwork embeddings. This effect is reinforced for larger hypernetwork sizes, see Figure 1.

Table 2: 5-way Few-shot classification accuracy (%) on MiniImagenet. For comparison with larger models, we include results for CAVIA with 3 different channel sizes for the convolutions used in the network. See [6] for more details. Mean $\pm$ std. over 5 seeds[1].

Method	1-shot (↑)	5-shot (↑)
MAML [1]	$48.07^{\pm 1.75}$	$63.15^{\pm 0.91}$
CAVIA$^{(32)}$ [6]	$47.24^{\pm 0.65}$	$59.05^{\pm 0.54}$
CAVIA$^{(128)}$	$49.84^{\pm 0.68}$	$64.63^{\pm 0.54}$
CAVIA$^{(512)}$	$51.82^{\pm 0.65}$	$65.85^{\pm 0.55}$
REPTILE [24]	$49.97^{\pm 0.32}$	$65.99^{\pm 0.58}$
Meta-SGD [8]	$50.47^{\pm 1.87}$	$64.03^{\pm 0.94}$
LayerLR [7]	$50.55^{\pm 0.87}$	$66.64^{\pm 0.69}$
MC [7]	$54.23^{\pm 0.88}$	$68.47^{\pm 0.69}$
MH-C	$48.64^{\pm 0.33}$	$64.52^{\pm 0.51}$
MH	$49.41^{\pm 0.96}$	$67.16^{\pm 0.42}$
MH-O	$52.50^{\pm 0.61}$	$67.76^{\pm 0.34}$

5 Theoretical analysis on a toy problem

Recent work has shown that given a local smooth and convex optimization landscape, linearizing a network around some weights and then taking the second-order Taylor expansion of the loss function gives an accurate enough quadratic objective [25]. This motivates the study of noisy quadratic models (NQMs) as analytically-tractable simplifications of real-world problems [26]. In this following, we formulate a toy few-shot regression learning problem using a NQM and show analytically that an optimal linear hypernetwork trained with our proposed algorithm seeks to learn the underlying structure of the different tasks.

5.1 Problem Definition

Let $p(\mathcal{T})$ be a distribution of tasks over T, such that $\forall\, \mathcal{T}_b \in \mathrm{T}$, the task $\mathcal{T}_b$ is the optimization problem consisting in the minimization of the loss

$$\mathcal{L}_b = \frac{1}{2}(w - w_b^*)^T\, H_b\, (w - w_b^*) \qquad (9)$$

where $H_b \in \mathbb{R}^{N_w \times N_w}$ is the Hessian of the loss, $w \in \mathbb{R}^{N_w \times 1}$ the weight to optimize, and $w_b^* \in \mathbb{R}^{N_w \times 1}$ the target weight.

Table 3: Usefulness of feature adaptation for 5-shots on MiniImagenet. Results shown are measure in classification accuracy (%). Mean $\pm$ std. over 5 seeds.

Methods	Adapting	Frozen
MAML [1]	$63.15^{\pm 0.91}$	$61.50^{\pm 0.50}$
MH-C	$64.52^{\pm 0.51}$	$59.18^{\pm 0.49}$
MH	$67.16^{\pm 0.42}$	$46.21^{\pm 0.27}$
MH-O	$67.76^{\pm 0.34}$	$49.44^{\pm 0.71}$

To model a shared underlying structure across tasks, we make the assumption that the tasks share a common Hessian matrix, which we denote as H. Each task in T is therefore uniquely identified by its optimal weight vector w_b^*. We further assume the tasks $\mathcal{T}_b$ follow a distribution such that the optimal weight $w_b^* \sim \mathcal{N}(w^*, \Sigma)$ is Gaussian-distributed.

[1]Results for related methods are taken from the cited papers (and $\pm$ std therein).

We study a one-shot regression problem over T, and analyze the solution found by our proposed algorithm. This meta-learning problem can be seen as learning over a NQM, where each task is a Gaussian perturbation of the optimal weight vector w^* to which the model needs to learn to quickly adapt.

We consider the non-chunked, linear hypernetwork of the form

$$w = \theta(\alpha_0 + \alpha) \tag{10}$$

where $\alpha \in \mathbb{R}^{K\times 1}$ is the task-specific embedding initialized at 0 at the beginning of a task and adapted in the inner loop, $\alpha_0 \in \mathbb{R}^{K\times 1}$ and $\theta \in \mathbb{R}^{N_w \times K}$ the hypernetwork parameters learnt in the outer loop.

5.2 Hypernetworks learn underlying task similarities

Fast adaptation in the given context is measured on the number of steps necessary to minimize (up to a given error) the loss on a given new task. We begin by computing the gradient of the task loss w.r.t. the hypernetwork embeddings for a task $\mathcal{T}_b \in \mathrm{T}$. Given

$$\mathcal{L}_b(\theta, \alpha_0, \alpha) = \frac{1}{2}(\theta(\alpha_0 + \alpha) - w_b^*)^T H\,(\theta(\alpha_0 + \alpha) - w_b^*) \tag{11}$$

we have

$$\frac{\partial \mathcal{L}_b}{\partial \alpha} = \theta^T H(w - w_b^*) \tag{12}$$

resulting in the following change of b when taking a gradient step

$$\Delta w = w - \theta(\alpha_0 + \alpha - \gamma \frac{\partial \mathcal{L}_b}{\partial \alpha}) = -\gamma\,\theta \frac{\partial \mathcal{L}_b}{\partial \alpha} = -\gamma\,\theta\theta^T H\,(w - w_b^*). \tag{13}$$

We observe that if $\gamma\,\theta\theta^T = H^{-1}$, then we have $\Delta w = -(w - w_b^*)$, leading the weight to perfectly solve the new task in a single step. In such case, the matrix θ acts as the optimal preconditioner for the task embeddings α, allowing for fast adaptation to new tasks. The remainder of this section outlines how the hypernetwork can implicitly learn this preconditioner when meta-learning over T.

During training, the task $\mathcal{T}_b$ sampled for the inner loop is used in the outer loop optimization as well. The choice of reusing the same task in both loops is crucial for our algorithm to learn the common structure of the tasks (see appendix for further justification).

Given a task, the outer loop loss $\tilde{\mathcal{L}}_b$ is therefore the same as the inner loop loss $\mathcal{L}_b$, but evaluated with the updated task embedding after a single step, $\alpha = -\gamma \frac{\partial \mathcal{L}_b}{\partial \alpha}$:

$$\tilde{\mathcal{L}}_b = \mathcal{L}_b(\theta, \alpha_0, -\gamma \frac{\partial \mathcal{L}_b}{\partial \alpha}) = \frac{1}{2}(\theta\,\alpha_0 - \gamma\,\theta \frac{\partial \mathcal{L}_b}{\partial \alpha} - w_b^*)^T H\,(\theta\,\alpha_0 - \gamma\,\theta \frac{\partial \mathcal{L}_b}{\partial \alpha} - w_b^*) \tag{14}$$

$$= \frac{1}{2}\Big((I - \gamma\,\theta\theta^T H)(\theta\,\alpha_0 - w_b^*)\Big)^T H\Big((I - \gamma\,\theta\theta^T H)(\theta\,\alpha_0 - w_b^*)\Big) \tag{15}$$

In the outer loop, the hypernetwork weights θ, α_0 are optimized based on the stochastic validation loss $\tilde{\mathcal{L}}_b$ such that the expectation of the gradient over task distribution reaches 0 i.e. when

$$\mathbb{E}_{\mathcal{T}_b \sim p(\mathcal{T})}\left[\frac{\partial \tilde{\mathcal{L}}_b}{\partial \alpha_0}\right] = 0 \quad \text{and} \quad \mathbb{E}_{\mathcal{T}_b \sim p(\mathcal{T})}\left[\frac{\partial \tilde{\mathcal{L}}_b}{\partial \theta}\right] = 0. \tag{16}$$

The system of equations (16) is solved when $(I - \gamma\,H\,\theta\theta^T) = 0$, i.e., when $\gamma\,\theta\theta^T = H^{-1}$ (see Appendix B for additional details). This shows that our meta-learned hypernetwork weights can recover the optimal preconditioner and therefore solve new tasks in a single step.

6 Conclusion

We showed that meta-learning a hypernetwork and adapting its embedding is a natural candidate method to create good few-shot learners. The performance of our method MH demonstrates that hypernetworks offer a good combination of feature acquisition and quick adaptation. MH is scalable and simple enough to be adapted to different model architectures and its effectiveness sheds light on the role of overparametrization in meta-learning.

Acknowledgements

Johannes von Oswald was supported by the Swiss Data Science Center (J.v.O. P18-03). João Sacramento was supported by an Ambizione grant (PZ00P3_186027) from the Swiss National Science Foundation.

Broader Impact

Fast adaptation and generalization on a wide range of environments is key to improve future artificial intelligent technologies. Although neural networks are highly flexible function approximators, they often do not generalize well to unseen tasks. In this work, we shed light on this problem and offer solutions to mitigate this problem. This line of research can have widespread impact in fields such as robotics, data analytics and artificial intelligence.

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, July 2017.

[2] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of MAML. In *International Conference on Learning Representations*, 2020.

[3] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. 2019.

[4] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B. Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? *arXiv preprint arXiv:2003.11539*, 2020.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[6] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. volume 97, pages 7693–7702, 2019.

[7] Eunbyung Park and Junier B. Oliva. Meta-curvature. In *Advances in Neural Information Processing Systems 32*, pages 3309–3319, 2019.

[8] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[9] Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2020.

[10] Jürgen Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 4(1):131–139, January 1992.

[11] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[12] Christoph von der Malsburg. The correlation theory of brain function. *Models Neural Netw.*, 2, 01 1994.

[13] Geoffrey E. Hinton and David C. Plaut. Using fast weights to deblur old memories. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, pages 177–186. Erlbaum, 1987.

[14] Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. In *International Conference on Learning Representations*, 2019.

[15] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

[16] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. *arXiv preprint arXiv:1709.07871*, 2017.

[17] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

[18] Vikranth Dwaracherla, Xiuyuan Lu, Morteza Ibrahimi, Ian Osband, Zheng Wen, and Benjamin Van Roy. Hypermodels for exploration. In *International Conference on Learning Representations*, 2020.

[19] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. *arXiv preprint arXiv:1803.10082*, 2018.

[20] Siddhant M. Jayakumar, Wojciech M. Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*, 2020.

[21] James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E. Turner. Fast and flexible multi-task classification using conditional neural adaptive processes, 2020.

[22] Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, 2018.

[23] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2016.

[24] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

[25] Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*, pages 8580–8589, 2018.

[26] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. In *Advances in Neural Information Processing Systems 32*, pages 8194–8205, 2019.

[27] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[28] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29*, pages 3630–3638, 2016.

Algorithm 1: Meta-learning via hypernetworks

Require: Task distribution $p(\mathcal{T})$, inner loop learning rate η, outer loop learning rate γ

Require: Hypernetwork-parameterized model $f_{\theta,\alpha}$ with randomly initialized hypernetwork weights θ and embedding weights α

while *not done* **do**

  Sample batch of tasks $T_B \sim p(\mathcal{T})$

  // inner loop

  forall *tasks* $\mathcal{T}_b \in T_B$ **do**

    $(\mathrm{D}_b^{\text{train}}, \mathcal{D}_b^{\text{val}}) \sim p(\mathcal{T}_b)$

    $\alpha_b \leftarrow \alpha - \eta \nabla_\alpha \frac{1}{|\mathcal{D}_b^{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_b^{\text{train}}} \mathcal{L}_{\mathcal{T}_b}\left(f_{\theta,\alpha}(x), y\right)$

  // outer loop

  $\alpha \leftarrow \alpha - \gamma \nabla_\alpha \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{D}_b^{\text{val}}|} \sum_{(x,y)\in\mathcal{D}_b^{\text{val}}} \mathcal{L}_{\mathcal{T}_b}\left(f_{\theta,\alpha_b}(x), y\right)$

  $\theta \leftarrow \theta - \gamma \nabla_\theta \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{D}_b^{\text{val}}|} \sum_{(x,y)\in\mathcal{D}_b^{\text{val}}} \mathcal{L}_{\mathcal{T}_b}\left(f_{\theta,\alpha_b}(x), y\right)$

Figure 1: Complete pseudocode for the proposed meta-learning via hypernetworks algorithm. For generality, each task $\mathcal{T}_b$ is allowed to have its own loss function $L_{\mathcal{T}_b}$.

Supplementary Material: Meta-Learning via Hypernetworks

A Experimental Setups

A.1 Few-shot Regression

We follow the same experiment described in [8, 1]. We perform few-shot regression on sinusoidal functions with amplitude and phase randomly sampled from [0.1,5.0] and [0,π]. In both the 5-shot and 10-shot setting, our meta model was trained on K simulated test datapoints, after training on K examples. At test time, similarly to [8], we obtain the MSE for one task by testing on 100 datapoints after the K-shot adaptation.

As outlined in [1],we used 1 gradient step with learning rate 0.0001 at training and test time in the inner loop. The outer loop was optimized with one gradient step using ADAM with learning rate 0.001. One training iteration consists of a batch of 25 tasks. The training phase ran for 70000 iterations and the best model was picked by early stopping on a held out validation set.

A.2 Few-shot Classification

We follow the same experimental setup as [1] on the MiniImageNet dataset. This dataset proposed by [27, 28] is separated into 64 training classes, 12 validation classes and 24 test classes. We used the same hyperparameters for the compressed,comparable and overparametrized model.The hypernetwork layers as well as the task embedding were initialized orthogonally. For the 1-shot and 5-shot settings, we used batch sizes of 4 and 2 respectively. In both settings, MH uses gradient steps with learning rate 0.05 in the inner loop and learning rate 0.001 in the outer loop. Both were trained with 6 inner loop gradient steps and tested with 15 gradient steps. The outer loop optimization was done with normal SGD. The best models for 1-shot and 5-shot classification were chosen with early stopping on a validation set and were trained for 100 000 and 150 000 iterations respectively.

A.3 Frozen Features in Inner Loop

All models were trained following the experimental setup described in A.2. For the frozen features setting, we follow [2] and fix the templates as well as the task embedding at test time. The only parameters updated in the inner-loop are the weights of the head. If MH was learning good embeddings, the frozen features experiment would show similar performance as the normal setting one (this is the case for MAML). We can see a drastic difference between the two, showing that the hypernetwork is really helping during the adaptation phase.

A.4 Meta-Curvature

Recently, it has been shown that meta-learning a (compressed) preconditioning matrix M to modulate gradients used in the inner loop yields state-of-the-art performance [7, 9]. Interestingly, Meta-Curvature does not affect

the model itself. Unlike MAML, this algorithm only adapts parameter gradients through a learnable matrix of meta-parameters:

$$\theta_b = \theta - \eta\, M\, \nabla_\theta \frac{1}{|\mathcal{D}_b^{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_b^{\text{train}}} \mathcal{L}_{\mathcal{T}_b}\left(f_\theta(x), y\right), \tag{17}$$

$$\theta \leftarrow \theta - \gamma\, \nabla_\theta \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{D}_b^{\text{val}}|} \sum_{(x,y)\in\mathcal{D}_b^{\text{val}}} \mathcal{L}_{\mathcal{T}_b}\left(f_{\theta_b}(x), y\right), \tag{18}$$

$$M \leftarrow M - \gamma\, \nabla_M \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{D}_b^{\text{val}}|} \sum_{(x,y)\in\mathcal{D}_b^{\text{val}}} \mathcal{L}_{\mathcal{T}_b}\left(f_{\theta_b}(x), y\right). \tag{19}$$

We note that the authors explore various interesting ways to construct the meta-parameter matrix M.

B Derivation for NQM model analysis

B.1 Same task used in inner and outer loop

Let $\mathcal{T}_b \in \mathrm{T}$. From (14), one can verify that

$$\frac{\partial \tilde{\mathcal{L}}_b}{\partial \alpha_0} = \theta^\top (I - \gamma H\theta\theta^\top) H (I - \gamma\theta\theta^\top H)(\theta\alpha_0 - w_b^*), \tag{20}$$

$$\begin{aligned}\frac{\partial \tilde{\mathcal{L}}_b}{\partial \theta} =& (I - \gamma H\theta\theta^\top) H (I - \gamma\theta\theta^\top H)(\theta\alpha_0 - w_b^*)\alpha_0^\top \\ &- \gamma H \Big[(\theta\alpha_0 - w_b^*)(\theta\alpha_0 - w_b^*)^\top (I - \gamma H\theta\theta^\top) \\ &+ (I - \gamma\theta\theta^\top H)(\theta\alpha_0 - w_b^*)(\theta\alpha_0 - w_b^*)^\top \Big] H\theta.\end{aligned} \tag{21}$$

By taking the expectation of these gradients over the task distribution, since w_b^* is sampled from $\mathcal{N}(w^*, \Sigma)$, we get:

$$\mathbb{E}_{\mathcal{T}_b\sim p(\mathcal{T})}\left[\frac{\partial \tilde{\mathcal{L}}_b}{\partial \alpha_0}\right] = \theta^\top (I - \gamma H\theta\theta^\top) H (I - \gamma\theta\theta^\top H)(\theta\alpha_0 - w^*), \tag{22}$$

$$\begin{aligned}\mathbb{E}_{\mathcal{T}_b\sim p(\mathcal{T})}\left[\frac{\partial \tilde{\mathcal{L}}_b}{\partial \theta}\right] =& (I - \gamma H\theta\theta^\top) H (I - \gamma\theta\theta^\top H)(\theta\alpha_0 - w^*)\alpha_0^\top \\ &- \gamma H(\theta\alpha_0\alpha_0^\top\theta^\top + \Sigma - \theta\alpha_0 w^{*\top} - w^*\alpha_0^\top\theta^\top)(I - \gamma H\theta\theta^\top) H\theta \\ &- \gamma H(I - \gamma\theta\theta^\top H)(\theta\alpha_0\alpha_0^\top\theta^\top + \Sigma - \theta\alpha_0 w^{*\top} - w^*\alpha_0^\top\theta^\top) H\theta.\end{aligned} \tag{23}$$

We can see that the expected gradients (22) and (23) both vanish when $\gamma\theta\theta^\top H = I$.

B.2 Different task used in inner and outer loop

When computing the outer loop loss on a different task than that used in the inner loop, the noise appearing in each of the loss are independent. By denoting by $\mathcal{T}_b, \mathcal{T}_d$, the independently sampled inner and outer loop tasks respectively, w_b^*, w_d^* their respective optimal weight vector, and $\tilde{\mathcal{L}}_{(b,d)} = \mathcal{L}_d(\theta, \alpha_0, -\gamma\frac{\partial \mathcal{L}_b}{\partial \alpha})$ the outer loop loss, equations (20) and (21) become

$$\frac{\partial \tilde{\mathcal{L}}_{(b,d)}}{\partial \alpha_0} = \theta^\top H (I - \gamma\theta^\top\theta^\top H)(\theta\alpha_0 - w_d^* - \gamma\theta\theta^\top H(\theta\alpha_0 - w_b^*)), \tag{24}$$

$$\begin{aligned}\frac{\partial \tilde{\mathcal{L}}_{(b,d)}}{\partial \theta} =& H(I - \gamma\theta^\top H\theta\theta^\top H)(\theta\alpha_0 - w_d^* - \gamma\theta\theta^\top H(\theta\alpha_0 - w_b^*))\alpha_0^\top \\ &- \gamma H \Big[(\theta\alpha_0 - w_b^*)(\theta\alpha_0 - w_d^*)^\top - \gamma(\theta\alpha_0 - w_b^*)(\theta\alpha_0 - w_b^*)^\top H\theta\theta^\top \\ &+ (\theta\alpha_0 - w_d^*)(\theta\alpha_0 - w_b^*)^\top - \gamma\theta\theta^\top H(\theta\alpha_0 - w_b^*)(\theta\alpha_0 - w_b^*)^\top \Big] H\theta.\end{aligned} \tag{25}$$

Taking the expectation of these gradients over the task distribution results in the following:

$$\begin{aligned}\mathbb{E}_{\mathcal{T}_b\sim p(\mathcal{T}),\mathcal{T}_d\sim p(\mathcal{T})}\left[\frac{\partial\tilde{\mathcal{L}}_{(b,d)}}{\partial\alpha_0}\right] =&\theta^\top H(I-\gamma\theta^\top\theta^\top H)(\theta\alpha_0-w^*-\gamma\theta\theta^\top H(\theta\alpha_0-w^*))\\ =&\theta^\top(I-\gamma H\theta\theta^\top)H(I-\gamma\theta\theta^\top H)(\theta\alpha_0-w^*),\end{aligned} \tag{26}$$

$$\begin{aligned}\mathbb{E}_{\mathcal{T}_b\sim p(\mathcal{T}),\mathcal{T}_d\sim p(\mathcal{T})}\left[\frac{\partial\tilde{\mathcal{L}}_{(b,d)}}{\partial\theta}\right] =&(I-\gamma H\theta\theta^\top)H(I-\gamma\theta\theta^\top H)(\theta\alpha_0-w^*)\alpha_0^\top\\ &-\gamma H\Big[(\theta\alpha_0-w^*)(\theta\alpha_0-w^*)^\top(I-\gamma H\theta\theta^\top)\\ &+(I-\gamma\theta\theta^\top H)(\theta\alpha_0-w^*)(\theta\alpha_0-w^*)^\top]H\theta\\ &+\gamma^2H[\theta\theta^\top H(\Sigma-w^*w^{*\top})(\Sigma-w^*w^{*\top})H\theta\theta^\top\Big]H\theta.\end{aligned} \tag{27}$$

We can see that taking independent noise in the inner and outer loop yields some covariance terms in (27) to be replaced by $w^*w^{*\top}$, resulting in an expected gradient which does not vanish in general when $\gamma\theta\theta^\top H = I$.